Porting Numerical Linear Algebra Libraries across Exascale Hardware Platforms and Beyond

Piotr Luszczek (speaker)

Hartwig Anzt*, Mark Gates, Stanimire Tomov

University of Tennessee, Knoxville; *Karlsruhe Institute of Technology

Scope: Libraries and Platforms

- Numerical libraries
  - Ginkgo
    - Sparse storage formats, iterative solvers, preconditioners, multigrid, SpMv
    - Variants: reference, OpenMP, accelerators
  - MAGMA
    - Dense: linear and eigenvalue/SVD
    - Sparse: iterative, preconditioners, storage
    - Mixed precision: 16-, 32-, and 64-bit solve
    - Variants: CUDA, clMagma, micMagma
  - PLASMA
    - Dense: linear, least-squares, EIG/SVD
    - Tile matrix layout and OpenMP 4 tasking
    - Variants: POSIX, WinThreads, OpenMP 4
  - SLATE
    - Distributed memory, multicore and GPUs
    - Flexible tile storage with affinity tracking

- ECP hardware/software
  - NVIDIA CUDA 10 and 11
    - Summit, Perlmutter
  - AMD HIP and rocM
    - Perlmutter, Frontier, Cray CCE
  - Intel DPC++
    - Aurora
- CI/CD systems
  - Cloud VMs and bare metal systems
    - Any modern x86-64
    - POWER
    - ARM

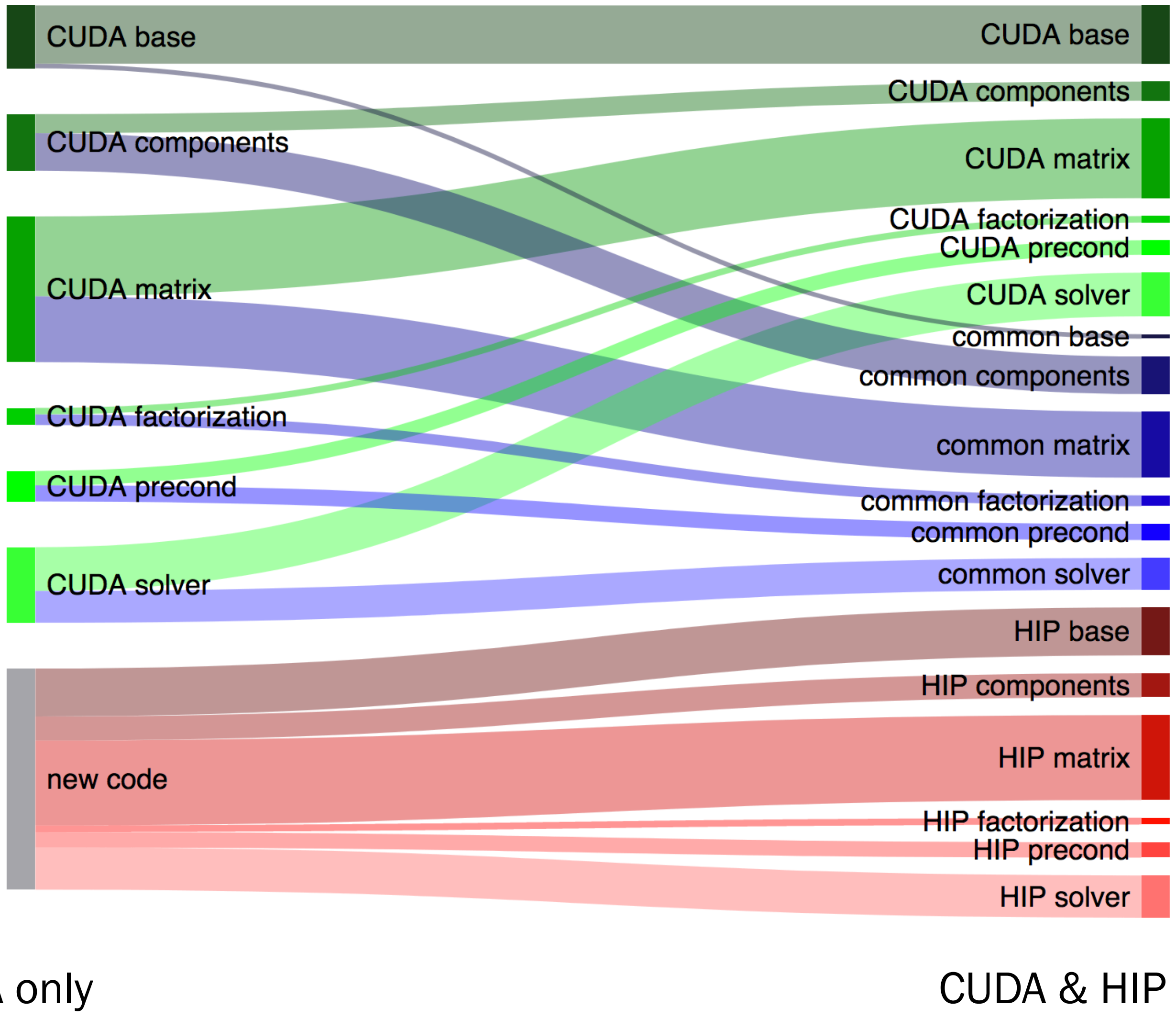
General Approach

- Common interfaces
  - Reuse or emulate established interfaces
    - DGEMM(), cuDgemm(), hipDgemm()
- Abstractions
  - Well defined and practical objects
  - Focus on user experience
    - Object hierarchy for matrix, vector, execution policy (device)
- Generic algorithms
  - Programming against generic types
  - Testing on concrete types

Use Case: Ginkgo

Ginkgo: from CUDA to HIP

- Kernels
  - Mostly shared between CUDA and HIP
  - Considered common
  - Abstracting away non-portable features
    - Cooperative groups
- Backend-specific optimizations
  - Always added for new backend
  - Must be maintained independently



Ginkgo Porting Remarks and Challenges

- CUDA and HIP are now relevant alternatives for Ginkgo functionality
- Similarities of HIP/CUDA syntax allow for significant sharing
  - Even for low-level implementations
- (In case of Ginkgo) compiling HIP to target NVIDIA devices has moderate effect on performance
- Comparable performance of CUDA/HIP across Ginkgo functionality

Use Case: MAGMA

MAGMA: dense and sparse linear algebra

- MAGMA’s scope exposed issues in HIP stack
  - Filed with AMD and fixed with HIP/rocM 3.5
- Automated approach necessary due to a large code base
  - Hippify’ing tools have their limits for MAGMA code base
  - Remaining issues fixed with automation (again: see code size and its changes)
- Performance results
  - Still work in progress across MAGMA, HIP, and rocM libraries
  - No clear winner or guidelines emerge yet

Performance Results: DGEMM on MI25 & MI50

hipMAGMA 1.0

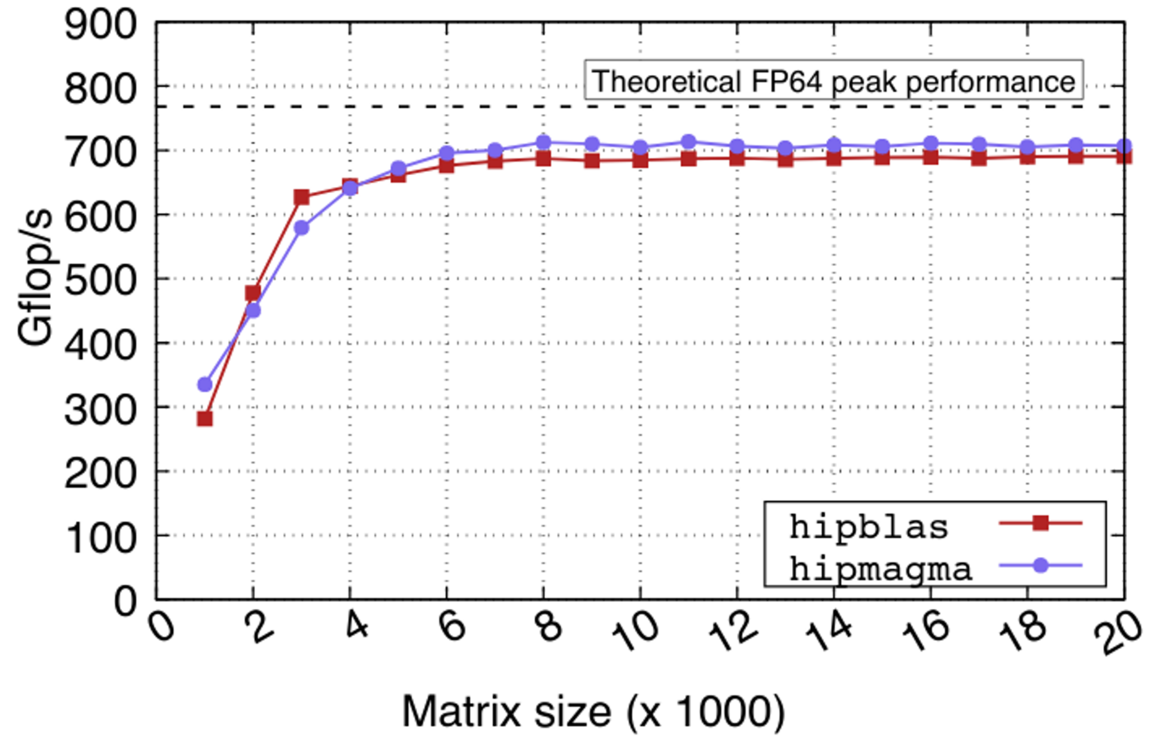


Fig. 3. Performance of the GEMM operation in double precision on the Mi2 GPU. Results are shown for square sizes using ROCm 3.5.

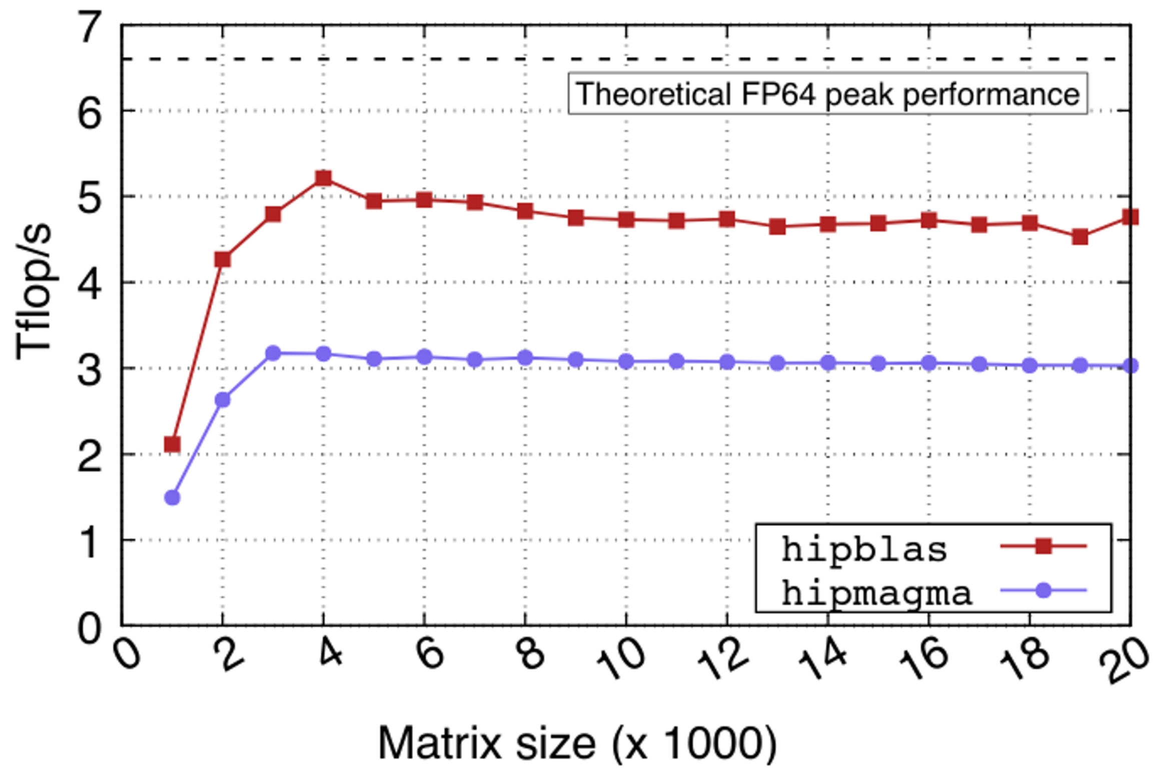
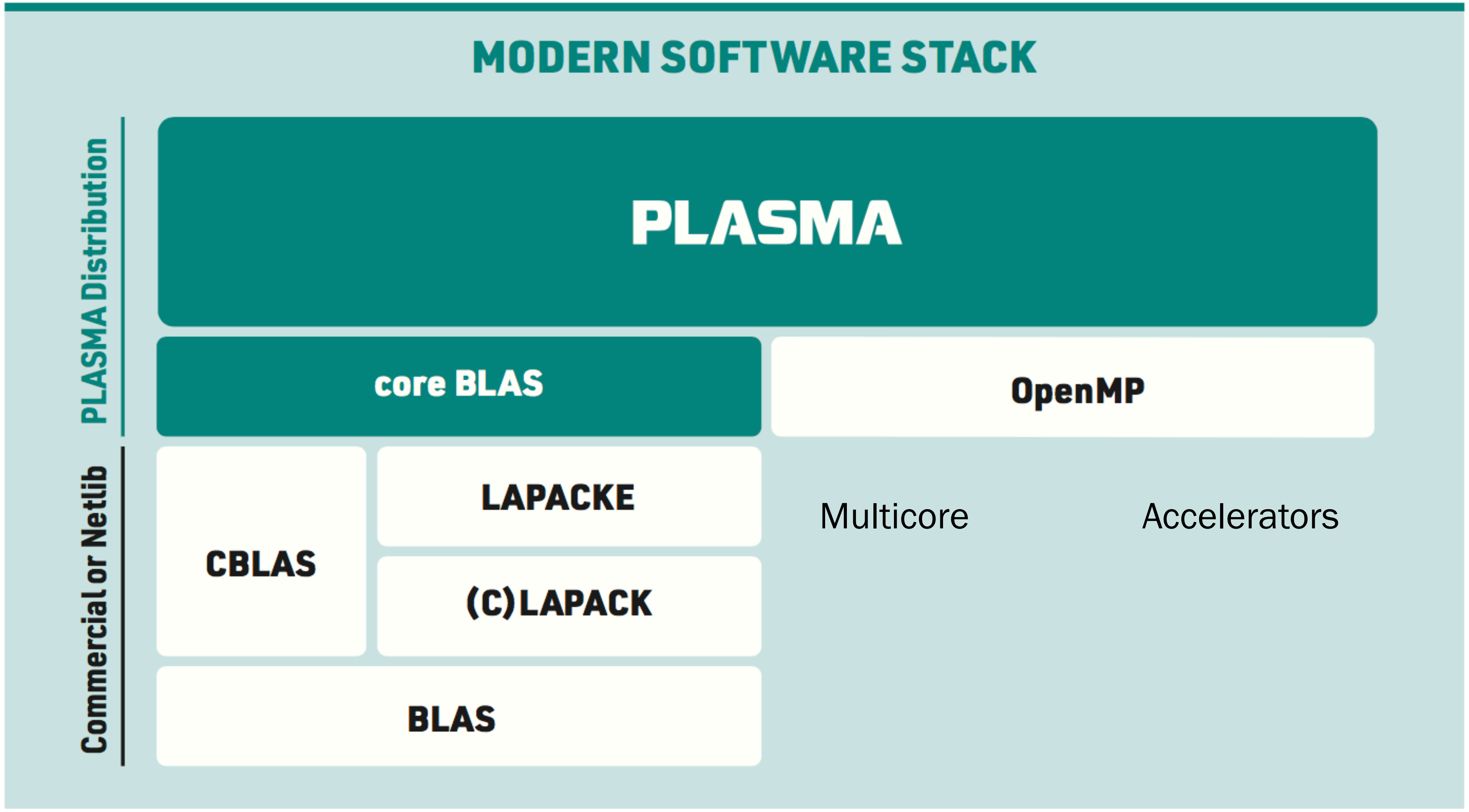


Fig. 2. Performance of the GEMM operation in double precision on the Mi50 GPU. Results are shown for square sizes using ROCm 3.5.

Use Case: PLASMA

PLASMA: dense linear algebra for OpenMP 4+



Use Case: SLATE

SLATE: dense and low-rank algorithms

- Large scope in terms of hardware
  - Multicore
    - ARM, POWER
  - GPUs
    - CUDA, HIP, DPC++
  - Distributed memory
    - MPI (multithreaded)
- Large algorithmic scope
  - Dense algorithms: linear, least-squares, eigenvalue, SVD
  - Matrix types and storages: rectangular, square, triangular, trapezoidal
  - Low-rank algorithms (ACA, low-rank tiles, …)
- Currently, work in progress